Henry Efor

Simulation and Optimization User Interface: A Flex Application

Helsinki Metropolia University of Applied Sciences
Bachelor of Engineering
Information Technology
Bachelor's Thesis
4 March 2011

Metropolia

Author Title Number of Pages Date	Henry Efor Simulation and Optimization UI (A Flex Application) 54 pages + 10 appendices 4 March 2011
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructors	Jaana Holvikivi, Principal Lecturer Taru Sotavalta, Language Instructor Jussi Rasinmaki, Instructor

The goal of this project was to explain the steps taken to create a Graphical user interface (GUI) for the SIMO forest management-planning framework. This was achieved using the Flex Flash builder. The SIMO UI was built as a client-server. The method used in the project was to devise an experimental model that would make it easy to build the GUI with fewer codes and still makes it flexible and easy to maintain in the near future.Mapping the XML directly with the components model was finally used since it is the only popular method used frequently in encoding and decoding XML documents. The results obtained showed that the Flex Adobe datagrid was not fully developed for the project and as such, I had to create a custom datagrid, the datasheet. Furthermore, as at the time of building the GUI, there was no credible way to work with namespaced XML. This problem was later solved through a series of research and writing custom-components to suit the application.In conclusion, the goal of the project was achieved, flexibility and maintainability, but at the expense of excess codes.

Keywords	SIMO, XML, FXG, MXML, RIA, VM, GUI, DOM, AVM, UI, UX

Contents

1 Introduction

In the summer of 2010, I started my internship with simosol Oy. Simosol Oy is a company established in 2008 that helps its clients in better decision-making about forests by building software that adapts to the needs of the client. The company is into the consulting and training of farmers so that they would increase their investment and reduce operation costs. In achieving these goals, I was faced with the task of building the User Interface of the simulation and optimization (SIMO) framework.

SIMO is a forest management-planning framework that helps the client to build different forest growth and yield simulators coupled with optimization methods and applying the combination to the management of forestry. The SIMO framework project was first developed at the department of Forest Resource Management at the University of Helsinki. The project is a joint one with other agencies such as UPM-KYMMENE LTD, Tornator Oy, Metsämannut Oy, and Metsähallitus Forestry Development Center Tapio. SIMO is an open-source project that is written in Python and as such, runs on various platforms as Windows, Linux and OS X.Simosol Oy is now the maintainer and main developer of SIMO.

The goal of this project is to explain the steps taken to create a Graphical user interface (GUI) for the SIMO forest management-planning framework. This will be achieved using the Flex Flash builder and Action script. The SIMO UI is built as a client-server. The server would be responsible for the management of XML documents by serving the XML documents to the client and will store them once the client returns the modified XML document. It will also do structural and content validation for the XML documents. The server part is outside the scope of this project. The project will concentrate on the client side development.

2 Theoretical Background

Before the steps taken to build the SIMO UI are explained, it is worthwhile to discuss the tool used in achieving the task. Simosol Oy wanted to use the same codebase for both web and desktop applications. In addition, the company wanted to do this quickly using a framework that already has a rich set of ready UI controls and components a framework that is extensible and design-friendly. There was only one framework that could do that – Flex. I had never worked with Flex but I soon realized that it is easy to use and it is convenient to build a Rich Internet Application (RIA) that can be deployed on both web and desktop. In this chapter, I will explain the Flex framework.

The Flex framework is a stack of classes that makes it easy to build rich-client side applications on top of the Action script implementation. Action script classes are compiled into byte code that runs within flash virtual machine (VM). The framework has already built-in containers, layouts, and controls that are all implemented in Action script. The framework gives the programmer access to the source code for the entire Flex framework and extend those components if the programmer has a special need since the source code is shipped with the Flex SDK. While any component can be built from scratch given the access to the source code, it is much like reinventing the wheel, as all that is there to do is to create an instance of any component and override any of the component methods.

2.1 Flex Overview

Animation metaphor was used to design the Flash Platform. It was challenging for traditional application programmers to understand and use the animation metaphor. This problem was solved with the introduction of Flex by Adobe. The programming model and the work flow that are characteristics of Flex made it easy for the programmers to understand and use it in Rich Internet Applications.

XML stands for Extensible Markup Language that comprises the rules that are to be followed to encode the documents in a form that is readable by the machine. An XML-

based markup language MXML was developed and introduced by Macromedia in March 2004. Graphic user interfaces are built with the help of this language. It is also used to make the layout of these interfaces. GUI is a user interface that enables the user to interact with the system through graphical symbols instead of typing on keyboard only. This phenomenon is used for exchange in computers, MP3 layers, gaming devices, and office equipments with images.

Client–side scripting on the web is done with the help of this language. The Flex SDK provides a wide range of user interface components that include buttons, trees, text controls, layout containers, list boxes, grids. Charts and graphs are also available. Other important features include web services, animation effects, modal dialogs, drag and drop function, form validation. The role of the Flex application is that of a presentation tier in a multitiered model. In the case of page-based HTML applications, a new page has to be loaded for every link on that page whereas in the case of the Flex application the client is not supposed to navigate to other pages. The links can be opened on the same page. The user does not have to reload the page to enable the Flex and Flash player to fetch fresh components of the page. Contemporary technologies include JavaFX, Ajax, XUL, OpenLaszlo, Curl and Windows presentation foundation. [4]

Different versions of Flex have been developed and introduced to the market. They include Macromedia Flex Server 1, 1.5, 2, 3, 3.5, 4.0,4.1 and 4.5(though this is still in beta phase). MXML and ActionScript are used for Flash applications.

In general, there are five steps involved in the Flex development processes given below.

1) By using a set of pre-defined components forms, button, an application interface is defined.
2) The selected components are arranged into the user interface design.
3) For the definition of visual design, the styles, themes and other visual factors are used.
4) Dynamic behaviors to be executed among different parts of the application are added.
5) In the final stage, the developed program is defined and connected to the data services as required.

The lifecycle methods available in Flex 3 are present in Flex 4 as well and perform the same functions as they were doing in Flex 3.

- CommitProperties: This method sets new properties.
- Update Display List: It is used to position and draw the subcomponents.
- Measure: This method is used to determine the size of the component.

Flex 4 by Adobe is loaded with a wide range of themes and applications that have made it useful to build RIA or Rich Internet Applications easily and quickly. Before going into the depth of these themes and utilities, it is important to know what RIA is in detail.

2.2 Rich Internet Application (RIA)

The term "Rich Internet Applications"(RIA) was coined by Macromedia. It was provided with the best practices in the user interaction design to make the use of Internet easier and attractive for people. Users showed extreme interest in RIAs since 2001 because of the utilities that are combined in them. These Internet applications are in extensive use especially by the consumer-facing sites. This technology is thought to be useful in internal and external business applications. Many enterprises and business setups are considering incorporating these technologies into their business. The RIA has many characteristics such as desktop applications. Commonly used platforms in RIAs are Java, Adobe Flash and Microsoft Silverlight. A software framework needs to be installed with the help of the computer's operating system to launch the application. The software is installed for the updates and downloads and it is essential for verification and execution of the RIA. Online gaming and applications with requirements of video capture primarily utilize the RIAs majorly. [3]

- RIA Benefits in Business:
    - Higher completion rates.
    - Increased customer loyalty.
    - Higher utilization rates.
- RIA Benefits for Developers:

- Ease maintenance.
- Less code.
- Multiple features.

- RIA Benefits in IT Operations:
  - Less server load.
  - Less bandwidth. [3]

2.3 ActionScript 3.0

In Flex 4, codes are written in MXML (xml files with a .mxl extension) and Action script. There is no Flex 4 without Action Script and MXML. ActionScript is an object oriented programming language belonging to the ECMAScript (ECMA-262) edition 4-draft language specification. It is open software that is deployed to specifically target Adobe Flash Player platform.

ActionScript 3.0 is a revolutionary, object-oriented programming language that is best suited for developing Rich Internet Applications. Though the earlier versions of ActionScript were very efficient, they cannot compete with the advances of ActionScript 3.0. ActionScript 3.0 is endowed with ease of development and superb performance that make it the best choice in the Flash environment for highly complex Internet applications, object-oriented, reusable code bases and large data sets. The users and developers can enjoy excellent output with ActionScript 3.0 as it is loaded with applications and contents that can target Flash Player.

ECMAScript is an international and standardized programming language that is used for scripting purposes. ActionScript 3.0 is based on this programming language. ActionScript Virtual Machine (AVM) is built into the Flash Player. The AVM is responsible for ActionScript execution. The AVM1 was an older version so now ActionScript 3.0 has introduced AVM2 that has multiplied the ActionScript 3.0 code execution to many folds. The new AVM2 machine is built into Flash Player 9. ActionScript 3.0 has two parts: the Flash Player API and the core language. The core language is used to define the building

blocks of the programming language whereas the Flash Player is made up of classes and it also provides access to the Flash Player-specific functions. ActionScript 3.0 is provided with innovations and capabilities to strengthen and speed up the Internet applications development process.

ECMAScript for XML (E4X) transforms the XML into native data type that simplifies the processing of XML. Also Display List API has made working with visual objects much faster and straightforward. ActionScript 3.0 is a strong blend of language aspects of ActionScript 2.0 and ECMAScript with enhanced functions and properties. A wide range of runtime exceptions for common errors is introduced in ActionScript 3.0. The type information is preserved in ActionScript 3. 0 at runtime and it can be utilized for other purposes as well. The Flash Player does the runtime type checking that results in system safety improvements.

The concept of a sealed class is introduced in ActionScript 3.0. A sealed class is provided with a fixed array of functions and properties that are defined at the time of compilation so that additional properties and functions cannot be added. In this way, a stricter compile-time checking is done that makes the program strong and robust. ECMAScript for XML (E4X) is now standardized as ECMA-357. To manipulate XML, this language offers a set of constructs. E4X reduces the amount of code needed as a result of which it can easily streamline the development of those applications that manipulate XML [1].

2.4 MacroMedia XML and MXML

Extensible markup language (XML) is a W3C standard for the markup of a text document. Though XML is not a language in itself, it basically consists of the rules that are to be followed in creating other markup languages. It can also be called a Meta markup language. Languages that use XML syntax are called XML Applications. It is employed in large areas of human activities, for example finance, travel, insurance or physics. for document sharing and data storage. An increasing number of software applications have XML files working behind the scene. These software applications include Macromedia

Flash, Microsoft Office and Apple iTunes. The XML is flexible and robust making it applicable in a variety of fields.

XML comprises four basic components that are given here.

1. A document that is marked in XML language.
2. An XML schema in which the elements and the rules to use them are defined.
3. Style sheets for the presentation instructions.
4. Parsers to interpret the XML document. [5]

MacroMedia Extensible Markup Language (MXML) is a UI markup language that is based on the XML. Macromedia introduced the language in March 2004. Though, the acronym MXML does not have an official meaning, for some developers it stands for "Magic Extensible Markup Language". The language is used with products like Adobe Flex in combination with ActionScript for developing Rich Internet Applications or RIAs. This language is used for the interface layouts in Flex. Other areas that can utilize MXML are Internet application behaviors and business logic implementation. It is used with the Flex server that is responsible for its compilation into standard binary SWF files. This compilation can also be done with the help of Adobe Flash Builder IDE and free Flex software development kit. Another package " XML_MXML" can also be used for building Adobe Flex Applications. This language is very well integrated with Adobe Technologies. MXML documents cannot be converted into any other user interface language e.g. XAML, SVG or XUL as there is no translator available for this purpose. [3]

Some MXML programming examples regarding the components, ID and properties of MXML are given below.

MXML- Component

```
<Application>
      <Button label="Get Data" click="ws.getProducts()"/>
       <DataGrid dataProvider="{ws.getProducts.result}"/>
</Application>
```

```
MXML - "id"
      <Application>
           <Button label="Get Data" id="getBtn"
```

```
            click="getProducts()"/>
             <DataGrid dataProvider="{ws.getProducts.result}"/>
      </Application>

MXML - Properties
      <Application>
            <Button label="Get Data"
            click="callBtnMethod()"/>
             <DataGrid dataProvider="{arrCollection}"/>
      </Application>
```

Listing 1: MXML properties and components
An example of MXML Component, Id, and Properties

3 Flex 4 Features

3.1 Themes

There were UI and UX integration requests from the users that were seriously considered by the Flex team when Flex 4 was under development. The team has made changes in the Flex framework. With the help of Flex 4, users can now enjoy better customization in RIAs. Flex 4 is provided with better and updated skinning and other properties.

The Flex framework and the Flex builder are able to strengthen the RIA. The Adobe team has put all its effort into enhancing the developer productivity in Flex 4. To improve the application development process, data binding and compiler performance have been updated. Also the automation support for Adobe AIR is added. Flex 4 is provided with ASDoc support for MXML documents. An additional CSS selector are also added so that the developers have multiple options and styling power for better customization of the look and feel of Internet applications that they make with Flex 4.

A wide range of enhancements and features has been added to the Flex 4 iterations that have made them more practical and strong. In addition to that, the Flash Player 10 capabilities can also be enjoyed which are added in this version. AIR 1.5 feature set is also added. Adobe Flash Detection Kit is replaced by widely used and open source SWFobject that will be used by the HTML publishing template. Although already present in Flex 3, Flex 4 is also provided with video component support. [4]

3.2 Spark Components

The Halo component is used to describe the component set in Flex 2 and 3. In Flex 4 the users are provided with the option to either use only Halo or both Halo or spark. Flex 4 utilizes Flash Player 10 capabilities and use base classes. Focus management and drag and drop functionality can be seen in Flex 4 as well, as in Flex 3. There is cross-compatibility between the spark and halo components so that the user can make use of both at the same time.

New base classes are added to Flex 4 to make it more useful and practical for the users. The classes are enlisted in the Flex class library.

- spark.components.supportClasses.SkinnableComponent

  This class defines the base class for the skinnable components. The skins that are used by this component class are the child classes of the skin class.

- spark.components.Group

  This is the base container class for the visual elements. The components that use the IUI component interface and the IGraphic Element interface are children for Group class.

  - spark.components.MXMLComponent

    The base class for this component is spark.

  - spark.components.supportClasses.core.Skin

    The skin class is considered the base class from all the skins that are used by the skinnable components and the skinnable component class is the base class for the skinnable components. [4]

A few new components are added to Flex 4. These components are in accordance with the rich and powerful skinning architecture of Flex 4.

- Spark Application Component:

  This component has introduced new layout capabilities in Flex 4. The default layout was vertical in Flex 3. It is now absolute in Flex 4. The layout is set through a tag now as is clear from the example given below.

```
<layout>
   <VerticalLayout />
</layout>
```

- Spark Button Component:

The spark button is a rectangular button that appears pressable. Any type of text label, icon or both can be adjusted on its face. It utilizes the skinning architecture. The look and feel for the Spark Button comes from the Skin class: ButtonSkin that utilizes MXML and FXG. [4]

3.3 Flash XML Graphics (FXG)

Scalable Vector Graphics (SVG) was primarily used long before FXG was introduced. SVG was just an XML-based graphic format. During the creation of SVG, it was not conformed to the Flex environment. So, a new MXML-based graphic tag was built. This is now what is known as FXG. For creating different shapes like paths, ellipses and other geometrical shapes, FXG is provided in Flex 4. The users can stroke and fill the shapes with colors, bitmaps and gradients with the help of FXG.

FXG is compatible with MXML. States and data binding can easily be used in FXG documents. FXG is loaded with features and capabilities. FXG enables the user to re-use multiple symbols within a single FXG document. Symbols can be styled in accordance with the graphical composition. Symbols contain graphic grouping elements that define individual graphical objects. [6]

3.4 States

In Flex, a state is the solution that makes the application dynamic. In order words, a state allows a change from one component to another in response to what state such a

component belongs to. States, while they already existed in Flex 3, are much easier to work with in Flex 4. [19]

The currentState property is used to track and manage the state within components and within the whole application. The includein and excludein properties are used to tell which of the states should appear during certain UI-display. The skin classes declare these states. Skin classes provide the facility to define multiple states that convey various meanings that are in accordance with the user requirements. [13]

3.5 Groups and Layout

Although, they were formally called Boxes, the concept of group was introduced in Flex 4. A group consists of an array of items that the group manages. The layout within the groups can be modified at runtime and is declaratively associated. There are the Horizontal group and the Vertical group in Flex 4. These are part of the non-visual components of the Flex programme.

Flex 4 is endowed with greater control and flexibility for the layout. The groups use layout objects in Flex 4. The layout provides the users with multiple options and flexibility in design development. With the help of the enhancements and variations done in Flex 4, multiple layouts can be used, replaced and modified at runtime. All the deficiencies that were seen in the Flex 3 Halo components are removed in Flex 4. Additional support is also given to the new layout types that include Basic layout, Horizontal layout, Vertical layout etc. Custom layouts can also be created if the need arises.

3.6 Skins

The Skinclass property is used to reference skin classes. These classes contain root element: “skin”. FXG defines the user interface for the components that of the skin class.

Skin parts are composed of elements that combine to form a component. The skin parts though are declared in the component class, yet implementation and visual appearance are defined in the skin class.

3.7 Localization

A useful feature that was used in the project was localization. One of the requirements of the project was that The SIMO framework should target both Finish and English users. In order to achieve this requirement, localization was set in. Localization is the process of including assets to support multiple locales. A locale is the combination of a language and a country code e.g: en_US means English language as spoken in the United States while en_GB means English language as spoken in the Great Britain. [12][20]

To localize an application to be used in Finland and the U.S.A, two sets of assets are provided, one for the en_US locale and the other for fin_FIN locale. Locales can share languages. An example is the en_US and the en_GB, which are different locales but still, use the English language. So, the word localization is spelt that way in the en_US locale while it is spelt localisation in the en_GB locale. Currencies and date format also differ in both assets and locale.

Localization encompasses more than just word translation. It includes any type of asset such as images and videos. This is because of the different meanings of images in different places. For example the red colour could mean danger for a set of culture while it simply means something else for another set. In Flex, to localize a Flex application, properties file that define the localized assets are created. Here is a snippet of what the properties files look like for an English locale:

```
#en_US
title          = Parameter Table
newtable       = New Table
```

The property file is compiled into application as resource bundles or the developer can create resource modules from the properties files and load them at run time. There are also the issues of whether the application has one or two or more locales. If the application supports many locales, it is appropriate to load the necessary resources at run time instead of compiling all supported resources into the application at compile time. Doing this is simply compiling the resource bundles into resource modules. [13]

To determine which locale the users want the following methods are employed:

User prompt: The application is started in some default locale, and then the user is asked to choose their preferred locale. This is actually the method used in the project.

Accept-Language header: In this method, a list of languages is sent to the server in the Accept-Language header when a browser makes an HTTP request. This header is then parsed in the HTML wrapper and passed as a flash variable into the flex application.One problem is if the developer has used an URLLoader to load an application, access to this header is denied.

In figure 1, a directory structure for a localization property file is shown:

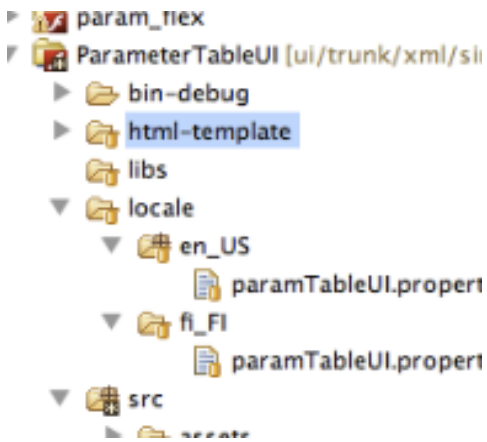


Figure 1: localization properties file directory

Immediately after creating the locales, the locales are then added to the compiler options of the project this way: -locale = en_US, fi_FI. Then the framework for the locale is included. For the en_US locale, this is exceptional as it is already added by default. To create a locale's framework resources, use the copylocale utility in the /sdk/bin directory. For Flex Builder, the copylocale utility is located in flex_builder_install/sdks/4.1.0/bin. You can only execute this utility from the command line. The fi_FI framework locale can be created by issuing the command: copylocale fi_FI on the command line.

The resource manager is used to get the locale or sometimes called resource bundles after creation and addition of the locale to the compiler option. Under the metadata tags, the resourceBundle property is used to store the name of the locale.

```
    <fx:Metadata>

[ResourceBundle('paramTableUI')]

</fx:Metadata>
```

Listing 2, ResourceBundle metadata tag

After this, the resouce bundle is registered and is ready to be used. It can simply be used in any component by just calling the getString method of the resouceManager as in this code:

```
…

<Mx:DataGrid

          id="dGrid"
          width="100%"
          height="100"
          dataProvider =
    "parxml.getItemAt(0).classifiers.classifiers">
          <mx:columns>
               <mx:DataGridColumn
               headerText="{resourceManager.getString('paramTable
               UI','name')}" dataField="variable"/>
               <mx:DataGridColumn
               headerText="{resourceManager.getString('paramTable
               UI','level')}" dataField="level"/>
          </mx:columns>
    </mx:DataGrid>
…
```

Listing 3: ResourceManager getString Method

A full example code is given on this link:
http://hillelcoren.com/2008/09/12/resource-bundles-in-flex-wo-lots-of-extra-code/

3.8 Events

Events are triggered when a user interacts with an application. In Flex, events are an important feature of the Flex application. Events are like reminders in an application. They

give dynamism to an application through interaction with different components within the application.

Events are generated by external input or by user devices. Events can occur without any direct interaction with the user such as when a component is completed or destroyed. Like every other event-driven programme, events can be handled by adding an event handler. These are functions or methods that are targeted towards certain events. In flex, they are referred to as event listeners. The Flex event model is based on the Document Object Model (DOM) Level 3 events model. Although Flex does not adhere specifically to the DOM standard, the implementations are very similar. The event model in flex comprises the Event object and its subclasses, and the event-dispatching model. [21]

When a component is interested in gathering information about another component's object, it registers a listener with the component's object. When an event occurs, the object dispatches the event to all registered listeners by calling the event handler.

Events can also be attached to another components. [11] This is called event listener in Flex. When a component needs to get information about certain information, it attaches an event listener to the component it is interested in. To do this in Flex, the addEventListener () method is called on the component. A simple example would be if a certain component wants to know if the submit button of another component has been clicked. The addEventListener () method is called this way:

addEventLister (MouseEvent.click, callMethod). Where the MouseEvent.click is the event of interest. While the callMethod is the method that should be called when the click event is triggered. The click event is a built-in event and there are over 100 events in Flex. Flex also allows the developer to use custom events (events that are not built in). To create custom events, the developer has to create a class that extends the flex event class. The flash.events.Event class contains information about the events available in flash. An event object is created each time an event is fired and the object can be accessed to find out important things about the event. The event object is implicitly created and available in the background. In listing 4 below, an event is dispatched when a user clicks the display button.

```
<s:Application name = "EventObject"
xmlns:mx = http://wwww.adobe.com/2006/mxml>
<s:Script>
import mx.controls.Alert;
private function
handleClick(event:MouseEvent):void{Alert.show(String(event));}
</s:Script>
<s:Button label = "display" click = "handleClick(event)">
</s:Application>
```

Listing 4: A click handler

Every Flex application begins with the application tag. It shows the root origin of every flex programme/application. There are also three most important Flex tags:

- Declarations tags: All non-visual components come under this tag. As such, any visual component can be made non-visual by declaring such component under this tag.
- Script tags: This is where the action script code goes on an MXMLfile.
- Metadata tag: This tag encompasses all custom-components built by the user. It registers it so that the flex compiler identifies the component or property at run-time. [13]

There are other root tags but these three are the basic and common ones.

3.9 Architecture

Flex 4 is loaded with refinements and variations for rich Internet applications as a result of which Flex 4 is in use by more people as compared to the past. The new skinning and component architecture that is provided in Flex 4 is called Spark. It works parallel to Adobe Flash Catalyst as it is provided with 20 new components and other primitives that enhance the overall output and feel of the Internet applications.

Figure 2 shows the Flex architecture.

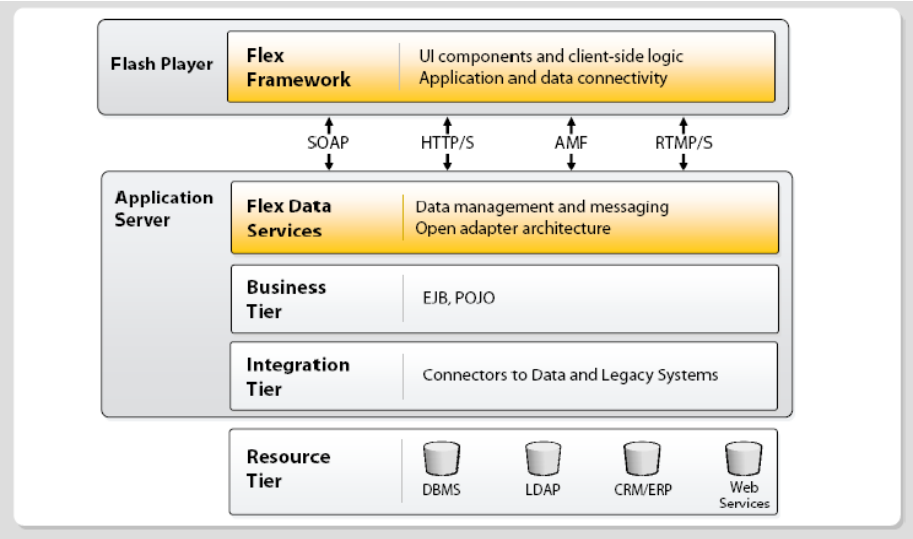


Figure 2: Flex architecture

[Copied from http://pshyama.wordpress.com/2009/06/12/adobe-flex-introduction-to-flex-3-part-1/]

4 Flex 4: Controls and Containers

4.1 Containers

Containers are structures that control the layout of a child component. Containers can be used to control the size and position of a given child and also to control navigation among multiple child containers. There are two types of containers namely layout and navigator. The spark containers used in Flex 4 contain controls and other containers. Some of the containers are canvas, controlBar, Form, Grid, HBox Panel, TitleWindow, VBox, Tile or TabBar.

Interchangeable layouts are supported by these containers so that the layout of any container can be set to any other layout type that is supported by the spark containers, such as vertical, horizontal, tiled layout etc. A custom layout can also be defined for fulfilling the individual requirements. Group and DataGroup containers can efficiently be used for managing child layouts. For custom skins support and child layout, skinnable data container and Skinable Container are used. [4][21]

4.2 Controls

Controls are visual components that mainly control the components. The controls are components placed on components. The root Flex application is the container, which contains all other containers and controls. Controls can be directly placed on the root Flex application or on other parent containers available in Flex. [13]

Most controls have the following characteristics:

- MXML API for declaring the control and the values of its properties and events
- Action Script API for calling the control's methods and setting its properties at run time.
- Customizable appearance by using styles, skins and fonts.

5 SIMO User Interface Development Cycle and XML Documents

This chapter describes the user interface development as well as the various XML structures involved in the development of the individual SIMO UI interfaces. The most detailed description of the XML documents are contained in the XML Schema files, which define the exact structure for the documents. The schema files are available through the SIMO development website:

http://trac.simoproject.org/browser/simo/trunk/simulator/xml/schemas [7]

SIMO framework consists of four main modules: data import, simulation, optimization and reporting, which are accessed using a command line interface (Fig 4) [18] .The components have been programmed with Python programming language, and the source code is available at http://trac.simo-project.org/browser. The main components implement a generic simulation and optimization framework based on the hierarchical data model; i.e., no forestry specific information is hard- coded into the framework components. All components use a key-value-based database (Oracle Berkeley DB) for permanent storage of serialized Python objects. Figure 4 below shows the SIMO modules.

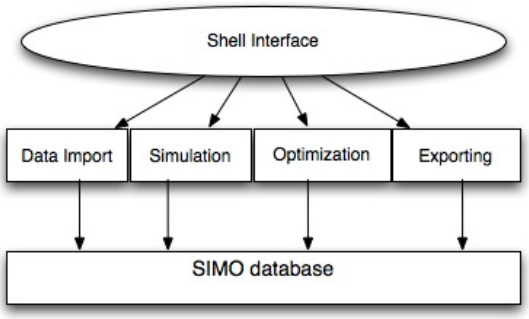


Figure 4. The SIMO modules

In an individual computation, the framework is modified using XML files that describe the precise content of the data model, and the simulation and optimization tasks for the computation. In addition, the programmatic implementations of the prediction and

operation models are needed. These are collected into model libraries, implemented as shared function libraries (dll-files in Windows, so-files in Linux/Unix).

There are over 10 XML documents in the SIMO framework:

Lexicon, Model Chain, Prediction Model, Operation Model, Aggregation Model, Management Model, Parameter table, Geo table, Cash flow, Cash flow model, Data conversion, Forced operation, Operation mapping, Operation conversion, Simulation, Optimization, Output Constraint, Aggregation definition, Expression definition, Lexicon translation, Message translation.

The idea behind my task was to build a user interface on top of these XML documents.

The task was to get these XML documents from the SIMO server and list them on a List Component using their name attribute. When a user clicks on any node on the search list, the UI area in the figure below is filled with elements from the data. The user then modifies it and the modified data is resaved to the correct node in the xml data and then stored in the server.

The general criteria are:

1. The user wants to edit an XML document of one of the SIMO XML document types.
2. UI requests the XML schema documents for the document type from the server.
3. UI requests the XML document from the server.
4. The XML document is converted into an ActionScript class instance with the help of the XML Schema document.
5. The UI component for the XML document type is populated with the data from the AS object created in the previous step.
6. The user modifies the AS object data using the UI
7. ActionScript (AS) object is converted back to XML document
8. The new, modified XML document is sent to the server

9. Localization of UI contents

The auxiliary criteria in some of the XML documents for example, the predictionUI, OperationModelUI, simulationUI are the use of the datasheet. Validation and Formatting are also requested in a few of the projects.

5.1 The SIMO DataSheet

The datasheet is a custom-made datagrid developed at Simosol Oy to give flexibility to a developer working heavily with inputs where each column should contain a strict validation of types. Suppose a company wants to create a payroll of its employees with a column containing the names of the employees and another column containing their ages. With the Flex dataGrid, one would have to use the validators and formatters, which is perfectly correct. Since there are just two columns, it is wise to keep it simple. Where there are three or more columns, it becomes logical to use the datasheet.

The datasheet allows a single datagrid to contain different types. On one column, can be XMLs and another is the int types or char type. The datagrid also comes in handy where a certain string should be tokened to various sub-strings. For example, in the parameter_table project, I had a task of representing the. <table>12345</table> as a datagrid containing five columns each containing the individual numbers that make up the <table>.</table> tag. With the datagrid, that is seemingly impossible because the datagrid property -dataField only takes a node from the source and displays it on the datagird. So, using the datagrid, one will have only one column containing a cell say A. In cell A is the string “12345”.

The problem can only be solved with the dataSheet as it allows mapping of objects in addition to explicitly passing in the block build, which is the default datatype of the datasheet. This default type can be changed using the objectClass property of the datasheet.

In the employee payroll analogy given earlier, it becomes relatively easy for such problems to be solved naturally. The user cannot enter a text where the descriptor

expects a numeric type. The datasheet has two important sub-classes, the rendererMap and the DataSheet descriptor. The rendererMap is responsible for the visual display of the cells while the descriptor, as the name suggest, describes the various types to be used on each column. The rendererMap is further divided into the cellMaps. The cellMap is responsible for how individual cells are displayed. It is divided into types, which contains the TypeMap. The TypeMap further contains the rendererFactory and the editorFactory.

The descriptor class describes the various types each column in the datasheet should adhere to. It is divided into the ComplexVarDescriptor, which is further divided into the properties classes (not shown in figure 3 below), which defines the characteristics of a certain complexVarDescriptor. There are basically five types of descriptors:

- TextVarDescriptor
- NumVarDescriptor
- CategoryVarDescriptor
- DateVarDescriptor
- ComplexVarDescriptor

The Figure 3 below shows the hierarchy of the dataSheet.

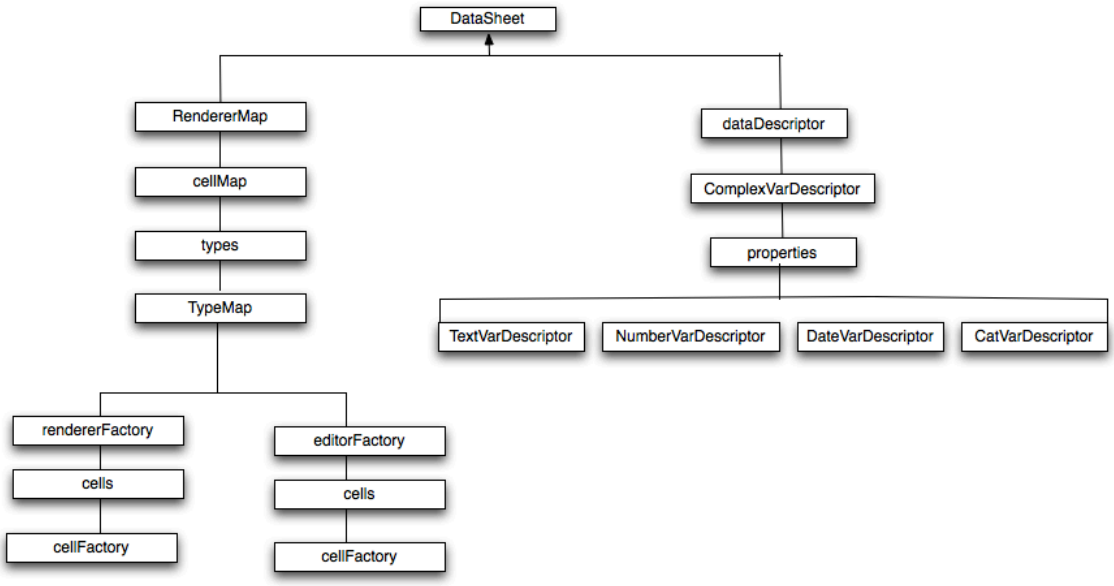


Figure 3: Datasheet class hierarchy

5.2 The Model

The model is responsible for requesting data from an external source and stores them until they are requested internally by the View component. There were three methods employed in developing the parameter table model. These are mostly experimental to see which model method is easiest and works for subsequent SIMO UI development.

1.Mapping XML documents to AS using XML Schema documents

2.Mapping XML documents to AS using a mapper library

3. Mapping XML documents directly as data source for the UI

5.2.1 Mapping XML Documents Using XML Schema Documents

XML Schema Document method involves using the XML Schema documents. [8] As I had just started working with the Flex environment, one of the initial tasks was to look for an existing solution that required minimal code input. A solution that will automatically convert the XML to Action Script as Value Objects (VOs) and vice versa. When I first looked into the Flex documentation, I only found Flex XML support for simple encoding and decoding. Little did I know that the Flex programme provides a wealth of encoding and decoding functionality for complex XML documents during a wild search? An article from an Australian web developer – Dominic De Lorenzo brought an end to the search. [8]

The mx.rpc.xml package of Flex 3 has XML Schema classes that can do both encoding and decoding of XML documents. A very good advantage of the XML Schema classes is that it allows a smooth registration of any ActionScript class definitions with element definitions defined within the XML Schema (XSD). This easy way to register Action Script class against elements helps to automatically serialize the XSD into ActionScript without populating the data.

One problem though is that the most XML Schema contains [ExcludeClass] metadata that excludes them from the documentation. The easiest way to get visibility of these classes in

your code is to place a copy of this package in your project and comment out all the [ExcludeClass] tags above the class definitions. The reason why Adobe has included the [ExcludeClass] will be explained at the end of this model type.

The key classes are:

1. SchemaManager (manages multiple schema definitions)
2. Schema (manages a single XML Schema Definition)
3. SchemaLoader (manages the loading of an XML schema (including any imports/includes) at runtime)
4. SchemaTypeRegistry (maps an XML schema type to an ActionScript Class)
5. XMLDecoder (decodes an XML document into ActionScript objects based on a XML Schema definition
6. XMLEncoder (encodes an ActionScript object into XML based of an XML Schema)

The primary steps are outlined below:

1. Create an instance of SchemaLoader and asynchronously load an XML schema from a given URL

This automatically loads any other schemas that are defined in XSD import or include elements.

```
schemaLoader = new SchemaLoader();
schemaLoader.addEventListener(SchemaLoadEvent.LOAD,
schemaLoader_loadHandler);
schemaLoader.load("example.xsd");
```

2. Once the schema is loaded, add it to the SchemaManager and register any ActionScript classes to their corresponding schema type:

```
privatefunction

schemaXMLLoader_loadHandler(event:SchemaLoadEvent):void
{
schema = event.schema;
schemaManager.addSchema(schema);
 schemaTypeRegistry.registerClass(new
QName(schema.targetNamespace.uri, "example"), ExampleVO);
}
```

3. Load an XML file based off that schema.

4. Once the XML is loaded, decode the contents using XMLDecoder. Any classes registered in the schemaTypeRegistry will be used when decoding the xml

```
var xmlDecoder:XMLDecoder;
var qName:QName;
var result:*;

xmlDecoder = new XMLDecoder();
xmlDecoder.schemaManager = schemaManager;

qName = new QName(schema.targetNamespace.uri, "results");
result = xmlDecoder.decode(xml, qName)
```

5. Encode a custom ActionScript class back into XML using XMLEncoder. XMLEncoder.encode() supports various ways to define the corresponding element in the schema (top level element, a specific type or even a custom XSD definition) that will be used to encode the Actionscript object.

```
var qName:QName;
var xmlEncoder:XMLEncoder;
var xmlList:XMLList;

qName = new QName(schema.targetNamespace.uri, "Example");
xmlEncoder = new XMLEncoder();
```

```
xmlEncoder.schemaManager = schemaManager;
xmlList = xmlEncoder.encode(exampleVO, qName);
```

For the full source code:
http://misprintt.net/examples/xmlSchema/srcview/index.html

There are a few factors to be aware of when using these classes.

1. It requires a XML Schema to already be in memory when decoding a loaded XML file. As this is an asynchronous task it is best to load the schemas required before loading any XML files based off them. The alternative is to check each XML file for schema information when they load, and delay the decoding of the data until that schema has then loaded. This would require additional checks and steps each time xml data is loaded.

2. Any additional schemas included or imported within the top-level schema loaded by SchemaLoader must be referenced using an absolute path. This is because SchemaLoader assumes that relative paths are relative to the main application swf URL (and not relative to the original XSD url). As these two locations are unlikely to be the same in a real world project, changing these URLs to be relative to the swf will make it work with Flex, but will prevent any other XSD editor from being able to find the corresponding includes or imports.

3. After a week of experimenting with this mapping type, I realized that the decoding does not work actively for a namespaced XML and the encoding does not encode correctly and completely the decoded XML. Like I said, prior to this mapping type that most of the functionalities come under the [ExcludeClass] metadata tag as a result of these flaws. It is there on their ASDoc as "TODO". It was for these reasons that I searched for another alternative to the XML Schema mapping type.

5.2.2 Mapping XML Documents Using A Mapper As Library

After the XML Schema was found not to support complex XML data structures, I discovered another mapping library that is used to map XML document called spicelib. Spicelib is a small AS3 library that can be used in Flex or pure AS3 projects. [9]
Spicelib contains a flexible encoding and decoding built in mappers which covers the most common use cases of mapping properties of AS3 classes to XML attributes or child elements. It can also be customized for some more complex XML elements. The metadata configuration support was added in version 2.3 making it relatively easy to use.
The only difference from the XML schema mapping method is that this method does not require a shema to function. Also, to validate an attribute or element, the [Required] metadata tag is applied to it while the [ChoiceType] metadata tag lists the permitted classes. The starting point is always to create the mapper as follows:

```
...
XML.ignoreWhitespace = true;
public var mapper:XmlObjectMapper = XmlObjectMappings
      .forUnqualifiedElements()
      .forNamespace("http://www.simo-project.org/simo")
      .withRootElement(ParameterTable)
      .choiceId("targets", Data, )
      .choiceId("Data", OperationParameter)
      .defaultSimpleMappingType(SimpleMappingType.CHILD_TEXT_
NODE)
      .mappedClasses(ParameterTables,ParameterTable,
Classifiers,Classifier
      Target, Targets, Value)
      . build();
```

Listing 5: spicelib mapper initialization

Note that this is a single line of code that extends afew lines down to the end-build ().
The next step is writing the decode and encode functions. The decode function looks something like this:

```
private function decodeXML():void{
      var xml:XML = ptXml.data as XML;
ptables = mapper.mapToObject(xml) as ParameterTables;
outputText.text = xml.toXMLString();
```

```
encodeXMLButton.enabled = "true";
}
```
Listing 6: Decoder

The Encoding is giving in Listing 8 below:

```
private function encodeXML():void{
      var xml:XML = mapper.mapToXml(ptables);
      outputText.text = xml.toXMLString();
}
```
Listing 7: Encoder

For a complete explanation of all these various methods and properties, a look at this link will help:
http://www.spicefactory.org/parsley/docs/2.3/manual/?page=xmlmapper&section=intro

The mapper mapping method could not be properly utilized because there were not enough examples to follow up the documentation and I had not the time to tweak around the spicelib library. There were also unclear informations about using this mapping method with complex XML data structure. At the moment, based on my work with this method, I believe the spicelib is still at its inception and further works and examples will make it complex XML data structure compliant.

5.2.3 Mapping XML Documents Directly As Data Source

A mapping method where each element in the XML node is directly tied to some component part of the UI. This mapping system works with the e4x format. [23] ECMAScript for XML (e4x) defines a set of classes and functionality for working with XML data. Afew useful methods are appendChild (), toString(), xmlToString(). ActionScript 3.0 contains embedded E4X classes that makes XML access in Flex easy and simple. In ActionScript 3.0, the XML class is called XMLDocument to avoid conflict with ActionScript 2.0, which also has an XML class.

In the XML structure given below:

```
<parameter_tables xmlns = http://www.simo-project.org/simo
           xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
           xsi:SchemaLocation =”http://www.simo-project.org/simo
           …/schemas/parameter_table.xsd”>
<parameter_table name="age_increment_forestland"
           type="parameter_table"
           desc="Age at breast height on forestland by
           municipality, site class and tree species">
           <classifiers>
                <classifier>
                     <variable>Municipal_id</variable>
                     <level>comp_unit</level>
                </classifier>
                     </classifiers>
           <targets>
                <target>
                     <data>
                          <variable>t_href</variable>
                          <level>stratum</level>
                     </data>
                </target>
           </targets>
                <parameter_table>
     </parameter_tables>
```

Using the E4X format, I could create an xml object that will return the classifier node in the parameter_tables xml using the ".” Operator by doing this:
var xml :XML = new XML(parameter_table.classifiers.classifier);

To get an attribute, the @ operator is used. For example, to point to the name attribute of the parameter_table xml, I could simply do this: xml.@name.

5.3 Parameter Table UI

In order to explain the project properly, I chose to use the parameter table UI as the model since the task is general for the other UIs. Where there are slight changes or deviation, I will explain it. I also chose the parameter table, as it was the first XML document I worked on and as such, there was ample experience gained that I wanted to

share in this project. The parameter table is modeled by mapping XML directly to the required components. The whole structure of the project is shown in figure 5 below:

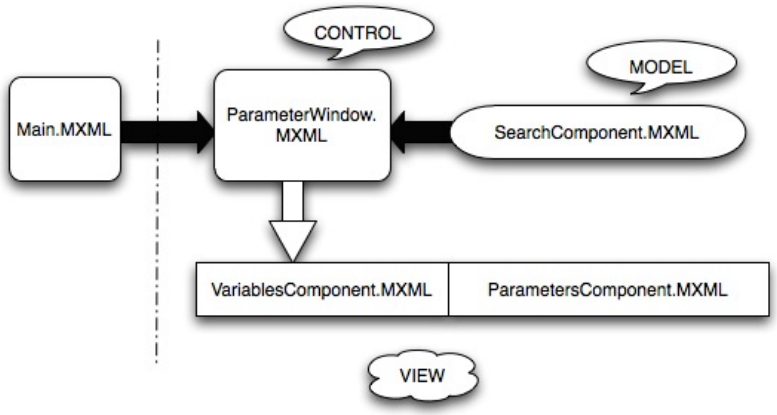


Figure 5. The parameter Table MVC

The whole concept was built around the Model-View-Control paradigm. The ParameterWindow acts as the control while the searchComponent acts as the model that gets the data and sends it to the parameterWindow to disperse the information/data to the views (variablesComponent and ParametersComponent). The Main.MXML is just an inlet to the whole MVC circle. The model and the views are children of the parameterWindow as shown in figure 6 below:

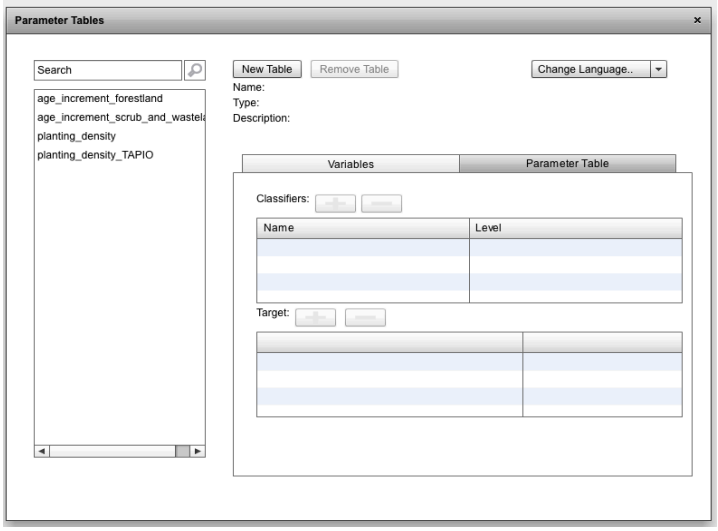


Figure 6: Search List and navigator component of the parameter table

The parameter table document defines parameter models used. An example of the document content is given below, for a comprehensive description of all the possible content can be found at:

http://trac.simoproject.org/browser/simo/trunk/simulator/xml/schemas/parameter

table.xsd [7]

The XML structure below defines a single parameter table node of the parameter _table XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
     <!--Copyright (c) 2007 Simosol Oy. Distributed under the GNU
     GeneralPublic License 2.0.-->
<parameter_tables xmlns = http://www.simo-project.org/simo
          xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
          xsi:SchemaLocation =”http://www.simo-project.org/simo
          …/schemas/parameter_table.xsd”>
<parameter_table name="age_increment_forestland"
          type="parameter_table"
          desc="Age at breast height on forestland by
          municipality, site class and tree species">
          <classifiers>
               <classifier>
                    <variable>Municipal_id</variable>
                    <level>comp_unit</level>
               </classifier>
               <classifier>
                    <variable>SC</variable>
                    <level>comp_unit</level>
               </classifier>
               <classifier>
                    <variable>SP</variable>
                    <level>stratum</level>
               </classifier>
          </classifiers>
          <targets>
               <target>
                    <data>
                         <variable>t_href</variable>
                         <level>stratum</level>
                    </data>
               </target>
          </targets>
          <table>
               <val>109 3 5 8</val>
```

```
                <val>109 3 6 8</val>
                <val>109 3 7 8</val>
                <val>109 3 8 8</val>
                <val>109 3 9 8</val>
                <val>109 4 1 12</val>
                </table>
      </parameter_table>
      </parameter_tables>
```

Listing 8: The parameter table XML structure

The root level tag contains a reference to the schema document, which is used to validate the content of the XML document:

```
<parameter_tables xmlns = http://www.simo-project.org/simo
           xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
           xsi:SchemaLocation =”http://www.simo-project.org/simo
           …/schemas/parameter_table.xsd”>
```

The parameter table includes importantly classifiers. The name attribute of the parameter table: “age_increment_forestland” is used as an example. The parameter table has attributes Municipal_id classifiers from the comp_unit level:

```
. . .
<parameter_table name="planting_density"
      type="operation_table"
           desc="Planting density by main tree species and site
           class">
           <classifiers>
                <classifier>
                     <variable>REGEN_SP</variable>
                     <level>comp_unit</level>
                </classifier>
                <classifier>
                     <variable>SC</variable>
                     <level>comp_unit</level>
                </classifier>
           </classifiers>
. . .
```

Listing 9. A single Parameter table XML node

The target is data when the result of paramter table is value of data variable. The model gives the value of regeneration_diameter attribute in comp_unit level as a result:

```
. . .
<targets>
      <target>
            <operation_parameter>
                  <parameter>N</parameter>
                  <length>1</length>
                  </operation_parameter>
                  </target>
            </targets>
```

Listing 10. A target XML node

Under the table node, regeneration diameter for the certain attribute combinations is given. First three digits are classifiers and the last is the regeneration diameter:

```
. . .
<table>
      <val>2 5 1700</val>
      <val>2 6 1800</val>
      <val>2 7 1600</val>
      <val>3 8 1600</val>
      <val>3 9 1600</val>
      <val>3 10 1600</val>
</table>
. .  .
```

Listing 11. A table XML node

The targets can also be data as given in the node below:

```
<targets>
      <target>
            <data>
             <variable>t_href</variable>
            <level>stratum</level>
            </data>
      </target>
</targets>
```

Listing 12. A target containing data XML node

Figure 7 below gives a visual hierarchy of the parameter table:

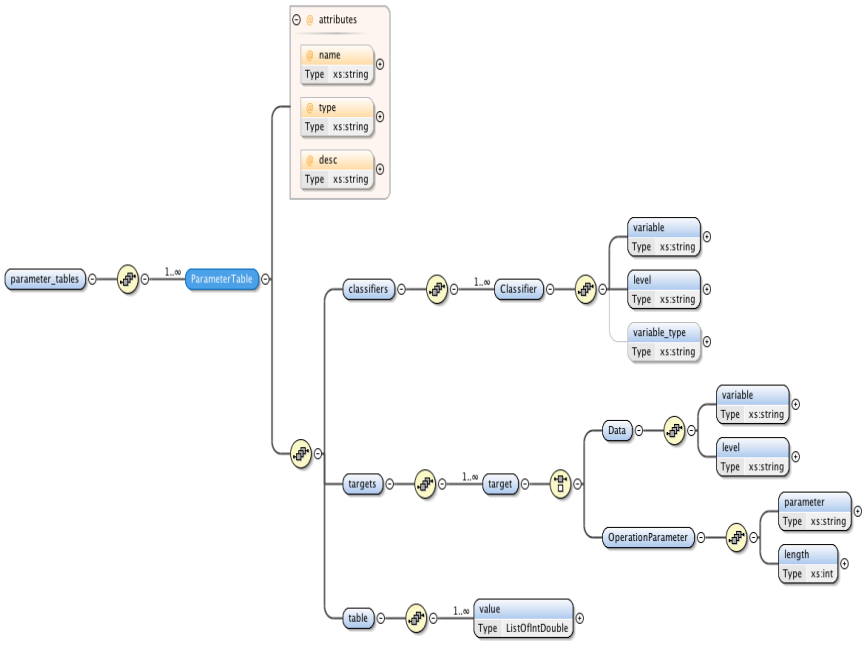


Figure 7: Hierarchy of the parameter table XML document

The parameter table UI programme contains a single source file, which contains the default file, assets, components, and skins. Then there are the locale file and the XML documents as well as the schema file for the parameter table. There are other folders but as said earlier, this was my first SIMO UI project, so, there are a lot of issues, I will be discussing as the process picks up momentum. The components folder contains the following files: ParametersComponent.mxml, ParameterWindow.mxml, ResultScript.as, script.as, SearchComponent.mxml, variablesComponent.mxml, SelectedTableEvent.as,utilityClass.as.

SearchComponent.mxml

The Search Component is a "VGroup" component containing the List Component where the XML nodes are populated. To populate the List Component, the XML

data has to be transferred from the server to the flex framework using two methods in flex, HTTP request or the XML loader method. In this project and in subsequent projects, the XML loader was used. To load the XML, the source property of the XML is used and the XML component is placed in the Declarations tag as shown below:

```
<fx:Declarations>
<fx:XML source = "../parameter_table.xml" id = "sxml" format
= "e4x"/>
</fx:Declarations>
```

What this code does is simply point to the source or location of the XML in the file hiearacrchy. In this case, the XML document is not stored in the source folder but under the parameter table project and with these three lines of code, the whole XML is now represented as an instance of the XML tag with object name "sxml". The next step is to now populate the List component with this XML. Again to do this is simple in Flex as shown:

```
…
<s:List id ="tableList" labelField = "@name" width ="190"
height = "430"
change = "tableList_changeHandler(event)">
<s:dataProvider>
<s:XMLListCollection source = "{sxml.children()}"/>
</s:dataProvider>
</s:List>
…
```

Listing 13: Populating a List component using the dataProvider MXML tag

The code has a list component with an instance name called tableList. In flex, each component is a class with properties and methods. So, the List component is a class with properties as labelField, width, height, change e.t.c The labelField, informs the compiler which property of the XML should be listed, from the code, it is the attribute names of each parameter nodes in the parameter_table .xml document. The @ sign signifies an attribute. IF there is"@description", that means, the XML has attribute property. In the parameter_table.xml, each

parameter-table node has a name attribute. This is the attribute with which each will be identified with. The change property dispatches an IndexChangeEvent event. This populates the different components of the Parameter UI. In the List component is the dataProvider. The dataProvider is simply a stream for all IList components in flex. It is the final container where external data are stored. [14] The XMLListCollection is the source of the dataProvider, which in this case is a list of XMLs displayed using their name attributes.

The next task is to map directly elements from the various nodes with the UI components. Now, that the XMLs are listed via their name attributes on the searchList, I proceeded to mapping them on both the variables component and the parameters component. Before then, there is still one more discussion under the searchcomponent.mxml. In the List component, is the change property that dispatches a MouseEvent using a handler called tableList_changeHandler(event). The tableList_changeHandler populates listens for a mouse click on any of the XML listed on the list component and populates the parameters components.mxml and the variable components.mxml with the elements of that particular XML node clicked. Since, a certain XML node on the XML lists should be mapped, I had to create a custom event to dispatch only that clicked XML. I first wrote an Event class called the SelectedTableEvent.as which contains the following lines of code:

```
Package components
{
import flash.events.Event ;
public class SelectedTableEvent extends Event;
{
public var xmlNode: XML;
```

Note the var keyword before the xmlNode. In Actionscript, variables are assigned that way using the accessor and then the var keyword after it.

```
public function SelectedTableEvent(type: String, xmlNode:XML,
bubbles:Boolean = false)
```

```
      {
      super();
      this.xmlNode = xmlNode;
      }
     }
}
```

Listing 14: Creating a custom Event

The code simply takes the selected xml as the second parameter of the SelectedTableEvent and initializes that in the constructor. Now, in the tableList_changeHandler(event) handler, I had to register my custom event using the [Event] metadata tag with this MXML code:

…

```
<fx:Metadata>
[Event (name = “itemClick”, type =
“components.SelectedTableEvent)]
</fx:Metadata>
```

…

Listing 15: The event metadata

Again, the name parameter gives the event a name and the type parameter informs the flex compiler where to look when responding to an event with that name. Finally, I could now dispatch my event in the tableList_changeHandler:

…

```
Protected function
tableList_changeHandler(event:IndexChangeEvent):void
{
var eventObject :SelectedTableEvent = new
SelectedTableEvent(“itemClick”,tableList.selectedItem,true);
dispatchEvent(eventObject);
}
```

…

Listing 16:dispatching an event

An instance of the SelectedTable Event class is created and that instance is dispatched.The dispatched event is sent to the SearchComponent of the

ParameterWindow.mxml where it is called properly. Here is how it is then called in the search component of the parameterWindow.mxml:

…

```
<components:SearchComponent
      id =”SearchWindow”
      includeIn =”initial”
      itemclick = “searchWindow_itemClickHandler(event)”/>
```

…

Listing 17: Event Handler of the search Component.mxml

As in the change property in the list component of the search component.mxml, the itemclick-the name given to the custom event when it was created- becomes a formal property of the searchComponent.mxml via the [Event] metadata and as such we can dispatch the searchWindow_itemClickHandler(event).Note, the searchWindow_itemClickHandler is a user –given name. The name of the event does not matter but as always, with the help of code completing, the Flex framework will always form this name using the id name of that component and appending the event name and finally a Handler_itemClickHandler to it unless the developer refuses to seek the help of code completing.
So, the searchWindow_itemClickHandler = id+eventname+_Handler(event) where the id is searchWindow, event name = itemclick .

ParameterWindow.mxml

The next component is the ParameterWindow.mxml component. This is the component that binds the Search Component, Variables Component and Parameters Component. The parameterWindow.mxml acts as a controller. Getting Information from the Search Component and passing it over to be displayed on the Variables and Parameter Components. It is also for this reason that the event

dispatched by the Search component is sent to the parameter Window component so that the variables and parameters components can be informed of the changes made on the search component.

It would be unwise to have to create all the various components on all three Components (Search, Parameter and Variables) on the Parameter Window.mxml. There must be a way to create these individual components separately and just create instances of each on the Parameter Window.mxml components. The code fragment below shows how those individual components are intermeshed into the parameter Window.

```
. . .
<components:SearchComponent

            id="searchWindow"
            includeIn="initial"
            x="10"
            width="30%"
            xmlList0= "{simocoll}"
            itemclick="searchWindow_itemclickHandler(event)"
            />
      <s:Group id="basicInfo" includeIn="initial"
      x="250" y="20"  height="496" width="508">

      <mx:TabBar
                  id="tabs"
                  dataProvider="{vs}"
                  y="120"
                  tabWidth="240.5"
                   paddingLeft="10" paddingRight="10"/>
            <mx:ViewStack
                  id="vs"
                  width="100%"
                  height="100%"
                  y="141"
                  paddingBottom="20"
                  borderStyle="solid">
            <s:NavigatorContent

<components:VariablesComponent
                              id="varcomp"
                              width="100%"
                              height="100%"
```

```
                                simoXml="{simoXml}"
                                left="10"/>

                        </s:NavigatorContent>

                        <s:NavigatorContent

<components:ParametersComponent
                                id="paramcomp"
                                width="100%"
                                height="100%"
                                itemCreationPolicy="immediate"
                                includeIn="initial"
                                simoXml="{simoXml}"
                                right="20"
                                />

                        </s:NavigatorContent>
                    </mx:ViewStack>
                    </s:Group>
                    . . .
```

Listing 18: ParameterWindow as a controller

There are three component tags. Each component tag comprises of the Search, Variables and Parameters. The tag "components" is not a flex component but the name of the file where the. mxml and.as files are stored. As a result, a namespace components should be declared at the beginning of the Parameter Windows like this: xmlns:components = "components.* " The other two components variables and parameters are embedded in another MXML tag called View Stack. This helps solely to navigate between which view is the current state (Flex states has already been introduced in Flex4: STATES). Figure 6 shown previously shows the visual display of the parameter Window with the variable component being the current state. Also, it becomes possible to refer to the variables, search and parameters components via their id (the id is like an instance of that class). So, with the "paramComp" id value of the parameters, the parameters components.mxml can be assessed from within the parameter Window components.mxml.

The Search, Variables and Parameters components act as models while the

Parameter Window is the controller that binds them all together in the itemclickHandler event handler. It should be recalled that the task was to model certain XML node in the Search component and the views of the variables and parameters are notified and populated via the parameter Window that is the controller. The parameter Window notifies these other two via the searchWindow_itemClickHandler (event) handler this way:

```
Protected function
searchWindow_itemClickHandler(event:SelectedTableEvent):void
{
1.var xmlNode:XML = new XML();
2.xmlNode = event.xmlNode;
3.varcomp.displayComp (xmlNode);
4.paramcomp.xmlNode = xmlNode;
}
```

Listing 19: event handler implementation

Here, the parameter clearly shows that the event is of the type SelectedTableEvent –the custom event created in the Search component and that event holds the currently clicked xml on the xml lists of the search component. This xml is passed on line 2 to a local xml variable created in line 1.

```
Var xmlNode:XML = new XML();
xmlNode = event.xmlNode;
```

Then, the display function of the variables components is called through its varcomp instance while the xmlNode property of the parameters is replaced with the xml from the event on line 4. Immediately after the programme exits this handler, the parameters and variables component both have copies of the updated node that was clicked on the search component list.

VariablesComponent.mxml

This component is responsible for displaying the XML document in a user-friendly way. It allows the user to modify and add new elements, or nodes to the XML data structure existing in the server.

Figure 8 below displays VariablesComponent view:

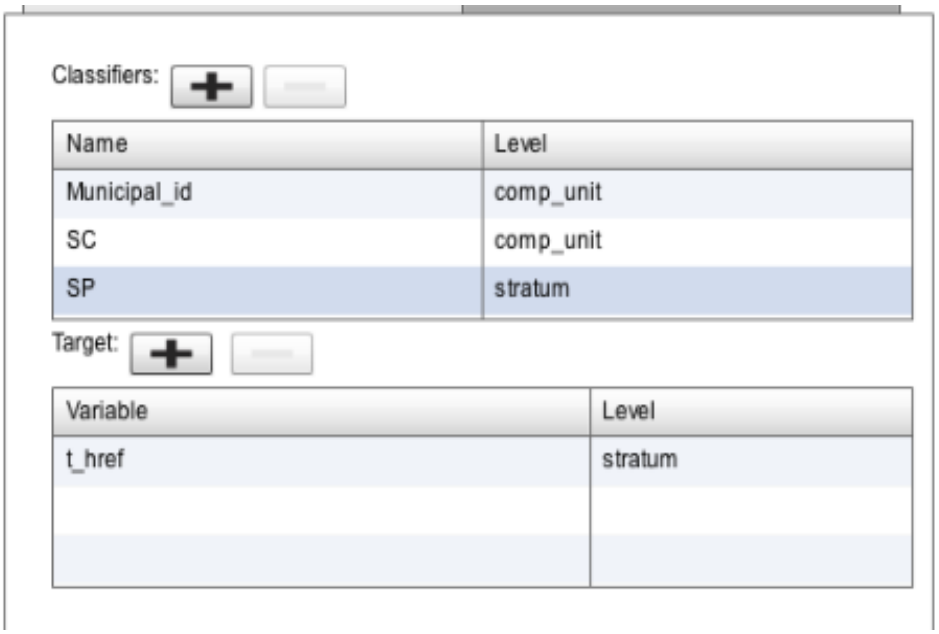


Figure 8: The variablesComponent view

In figure 8, there is the data part, which displays the data and operation nodes of the parameter_table XML data structure. Also there are two buttons (the add and remove buttons) up above the data part of the grid. There is also a second grid containing the table part of the XML data structure. The displayComp(xml) function passed the masterXml in the parameter Window.mxml displays the right elements in figure 6. The displayComp(xml) function does it by binding each component on the variables component.mxml to the XML data structure relating to that part as given below in listing 19:

```
public function displayComp(xmlNode:XML):void{
                var selecto:XMLList = new XMLList(xmlNode);
                var parxml:XMLListCollection = new
                XMLListCollection(selecto);
                dGrid.dataProvider =
                parxml.getItemAt(0).classifiers.classifier;
                addTarget.enabled = true;
                     addClassifier.enabled = true;
                var word:Array = new Array();
                var val:XML ;
                pdGrid.dataProvider =
```

```
                    xmlNode.targets.target.elements();
                    var str:String = xmlNode.toXMLString();
                    var type:String =
                    str.slice(str.indexOf("type")).split('"')[1].
              toString();
                    switch(type){
                         case "parameter_table":
                              currentState="data";
                              break;
                         case "operation_table":

    currentState="operation_parameter";
                              break;
                    }
              }
```

Listing 19: The display function of the variables Component

The variables component.mxml uses a data-driven component called the datagrid. The datagrid is a formatted table whose items can be edited and displayed through custom renderers. The features of a datagrid include: [13][22]

i. Resizeable columns

ii. Customizable columns and row hearders

iii. Editable

iv. Use of custom item renderer

v. Multiple modes of selection

The variables component.mxml datagrid component is populated with XML via the dataProvider of the datagrid. The dataProvider is a warehouse or storage place for data to be displayed by the datagrid. The code below in listing 20 shows how a datagrid is populated with the XML data using the dataProvider property of the grid. [18]

```
...
<Mx:DataGrid
          id="dGrid"
          width="100%"
          height="100"
          editable="{xmlNodeSelected}"
          dataProvider =
    "parxml.getItemAt(0).classifiers.classifiers">
```

```
            <mx:columns>
                  <mx:DataGridColumn
                  headerText="{resourceManager.getString('paramTable
                  UI','name')}" dataField="variable"/>
                  <mx:DataGridColumn
                  headerText="{resourceManager.getString('paramTable
                  UI','level')}" dataField="level"/>
            </mx:columns>
      </mx:DataGrid>
. . .
```

Listing 20: variablesComponent Datagrid

The dataProvider contains the data to be displayed. In this case, the dataProvider accepts only arrayCollection, XMLListCollection or some custom collections. The headerText property points to the headings of the table while the dataField points to the particular data to be displayed under a given headerText. Figure 9 shows a 4 x 2 datagrid with a headerText Name on row index 0 and column index 0 and under that column are the dataFields Municipal_id, SC,and SP. These are direct data from the parameter_table XML.

Name	Level
Municipal_id	comp_unit
SC	comp_unit
SP	stratum

Figure 9: variables Components datagrid.

ParametersComponent.mxml

The parametersComponent.mxml mxml component is the other view component in the parameter table XML GUI project. This uses the datasheet as its data-driven control. To use the datasheet, it has to be initialized. The datasheet can be initialized using the AS format or the MXML format. The choice of which to use depends on the validity of the data to be displayed. If on a single datasheet the descriptors are different, say certain

cells must contain numbers while another must contain text. The MXML format is better where each complexVarDescriptor is separately initialized but if the datasheet contains all round numbers or texts, then it is much easier to use the AS format where the complexVarDescriptor is ran in a loop. In the parametersCompoenent.mxml component, the datasheet contains all round numbers so, the AS format was used to initialize the datasheet. To initialize the datasheet, the following steps are taken:

1. First is to create the variable type. That is numbers or text.

```
private function varDescriptor(typeArr:Array,
numVarArr:Array, arr:Array):void
{
            var defaultMap:DefaultMap=new DefaultMap();
      var len:int=arr.length;
      for (var item:int=0; item < len; item++)
      {
            var num:TextVarDescriptor=new TextVarDescriptor();
            typeArr[item]=defaultMap.getTypeMap("text");
            num.name=arr[item];
            num.type="text";
            numVarArr.push(num);   }
```

Listing 21: Creating a descriptor

2. Then the variable descriptor is plugged into the complexVarDescriptor this way:

```
private function complexVarReg(dataArr:Array, typeArr:Array,
sheet:DataSheet):void
{
      var complexVar:ComplexVarDescriptor=new
ComplexVarDescriptor();
      complexVar.properties=dataArr;
      Next is creating the create CellMap instance, TypeMaps
to its types-Property.
      var cellmap:CellMap=new CellMap();
      cellmap.types=typeArr;
       CellMap to rendererMap property of DataSheet
      sheet.rendererMap=cellmap;
      ComplexVarDescriptor to dataDescriptor-property of
DataSheet
```

```
sheet.dataDescriptor=complexVar;}
```

Listing 22: Creating a complexVarDescriptor

After this, the datasheet can be used anywhere. For example, it is used in a display function of the ParametersComponents.mxml:

```
public function displayGrid(xml:XML):void
{
      var dataArr:Array=[];
      var temp0:XMLList = xmlNode.classifiers.classifier.variable;
      var labelArr:Array = new Array();
      var numVarArr:Array = new Array();
      var typeArr:Array = new Array();
           for each(var i:String in temp0)
          labelArr.push(i);
       varDescriptor (typeArr, numVarArr0, labelArr);
complexVarReg(numVarArr, typeArr, sheet);
Note, the rest of the code is just binding the various labels to
the data.
var amx:Array = new Array();
var obj:Object;
var mag0:int = labelArr.length;
var total0:int = dataArr.length;
for(var tl:int = 0; tl<total0;tl++)
{
obj = new Object();
for(var ni:int = 0; ni<mag0; ni++)
      {
      if(dataArr[tl][ni]=="-1")
      dataArr[tl][ni] = " ";
          obj[labelArr[ni]+ni] = dataArr[tl][ni

           }
     amx.push(obj);
}
sheet.dataProvider = new ArrayCollection(amx);
}
```

Listing 23: Function to display the datasheet

6 Discussion

When I started out working with the parameter table, I had no idea what the final GUI would look like. The plan was to get more informed on the overall task as I went along. In my opinion, while that approach helped me in understanding the various tasks involved in building the different GUIs, I did pay for it by having to recode most of the programme. There are times I had to rewrite the whole code or part of it due to a change in ideas. In this section, I intend to discuss problems I came across while working with flex and the GUI projects.

The first most important problem was to choose which model is best for decoding and encoding XMLs. As a result, the project started out searching for the easiest method to this problem.
These led to a series of trial and error models discussed in section 5.2. One major thing I discovered was that while the various examples and methodologies worked fine with simple xml structures, they did not support complex xml structures. The approach was a fast one where, if one method was not conducive, a shift to the next method was imperative. What I experienced while working with Flex built in encoder and decoder was that it optimized the project in that a few codes were required. In the future, I definitely will experiment more with the encoding and decoding when there is ample time at my disposal.

In the end, I had to settle for the conventional way of doing this – encoding and decoding the XML directly instead of using any optimized way. This proved to work well and produced positive results. Infact, in just two days, I was able to finish the parameter table and so far, it looks completely finished except that the parameter table XML document has been modified to include namespaces and in the future might still be subjected to further modifications. For data to be bound to the various component of the application, the namespaces had to be removed completely from the XML document. At first, having the

namespaces in the XML document made it impossible to bound the nodes or elements to any component and it took me a while to work with namespaced XML document. The solution to this problem became imperative as in the end, the original XML documents had in them namespaces. However, things were working fine now without the namespaces, but I still had to fix that problem. A look at the Action Script Documentation ASDoc on XML namespaces gave an insight into how to solve this problem. [13]

I have not corrected this namespace in the parameter table project. I discovered the solution to the namespace in a latter SIMO GUI project and I am sure I will still have to go back to the parameter table project to add those namespaces and do the necessary modifications needed. At the moment, suffice it to say the parameter table is near perfection. An easy way to solve this problem is to declare an instance of the NameSpace class and then use it as a default namespace in the whole project. After this, in every function where the XML will be bound to a given component, the keyword use namespace "NameSpace Instance" is first introduced. The example below gives a better understanding.

```
<?xml version="1.0" encoding="UTF-8"?>
<parameter_tables xmlns = http://www.simo-project.org/simo
           xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
           xsi:SchemaLocation ="http://www.simo-project.org/simo
           …/schemas/parameter_table.xsd">
<parameter_table name="age_increment_forestland/>
</parameter_tables>
```

It is a simple XML structure with just a single node called the parameter table but with namespaces as well. To bind the name element of the parameter_table to a spark text component, one has to declare the namespace:

Var nameSpace: NameSpace = new NameSpace ("http://www.simo-project.org/simo);

and then declare it as the default namespace.:"default namespace is namespace"

Also, one problem existing with building the model by mapping the XML directly to the flex components is that it requires some coding. The view also must adjust itself to the structure of the XML structure. In summary, an understanding of the meaning of the XML structure is required in using the direct mapping method of the XML.

The datasheet was not fully functional when I started off at Simosol Oy. It was built for a map project and as I was told, was abandoned somewhere along the line. I had to work with it and as such encountered some bugs. Prior to the operationModelUI project, it was a dead-end situation to work with the datasheet. However, I applied the datasheet to the parameter table project but it was just mainly to display the data. I really started working with other aspects of the datasheet both in the Action Script format and the MXML format in a later project where I discovered, that it was impossible to add to and to modify data on the datasheet. I also realized, everything had to be done by dispatching events, as the XMLListCollection was not supported. Objects had to be mapped to labels for display to be called.

The use of the datasheet was to eliminate the buttons and as such, the datasheet has to differentiate between when the user just wants to add data and when he wants to modify an existing data. It also has to know when the user deletes an item off the datasheet. So, at that stage, the datasheet was not doing all these and as such another two weeks of work on datasheet was ensured. In the end, the bugs were fixed and now, the datasheet is able to add, modify and delete data. As at the time I was writing this documentation, the datasheet has been extensively modified to accommodate XML collections. So, there was no need to do the regular binding that was shown in the displayGrid () function of the parametersComponent.mxml.

Furthermore, the Flex programming has its own naming convention and styles. A function name must begin with a small letter as in var func funcName () while a class name must begin with a capital letter as in NameSpace. An event class must begin with a small letter. In order to make the code readable and fit to these conventions and styles, I had to

rewrite the parameter_table at a certain stage to make it compatible with the standard naming convention and flex-programming style. As a C-programming programmer, I had also difficulty in adjusting to these naming conventions. There were also times when the visual appearance was not fitting to the current XML document structure of a particular project as in the operationModelUI, and it had to be rewritten to make it flexible and real.

7 Conclusion

The aim of the project was to build a flexible, maintainable SIMO GUI that would meet the demands of the client. Flexibility means the project is user-friendly and accessible while maintainability is the ease to maintain the project as more features are enhanced in the near future.

Since the parameter table was the model on which this whole project was built on, the project, while it met flexibility, did not meet the maintainability. At the moment, the project has been tested and it satisfies flexibility. The client can add, delete and update data on the UI and the server XML is updated as required. The user can also create a whole new parameter_table node to be appended to the parameter_tables root node.

However maintainability will probably be met when this project is carried out again. This feature could not be met as the whole parameter table project was based on experimenting with the tools available. In subsequent projects both flexibility and maintainability were met.

Also, the encoding of the XML document was not satisfied. The reason was that the server had to be ready for the encoding to be processed. Server-side programmers are currently working on this at Simosol Oy, and as a result, I will be resuming this project again when the server is ready for a proper and final optimization of all the various parts. Since the server side is not fully automated, all changes made to the UI are lost once the program is re-run again.

The project was not carried out in the MVC-fashion. The model and view were combined in a single file. That is something that could be modified in the future, such that it would make it possible for the code to be understood and maintained. In general, there is still much to be modified in the future as the SIMO GUIs project is an ongoing one and experimental.

References

1. Adobe. ActionScript 3.0. Article [online].
   URL:http://www.adobe.com/devnet/actionscript/articles/actionscript3_overview.html Accessed: 13th January 2011.

2. Adobe. Flex [online]
   URL: http://www.adobe.com/products/flex/
   Accessed: 13th January 2011.

3. Ashok M. Saravanan. Flex Basic [online]. Scribd.
   URL: http://www.scribd.com/doc/6153530/FLEX-Basics
   Accessed: 13th January 2011.

4. Sean Moore. Overview of Flex 4 (Gumbo) [online] DevelopRIA.
   URL: http://www.developria.com/2009/06/overview-of-flex-4-gumbo.html
   Accessed: 13th January 2011.

5. Robbins, J.N. Web Design in a Nutshell. O'Reilly Media Inc. CA. 1999.

6. Adobe (Open Source). FXG 1.0 – Functional and Design Specification [online].
   URL: http://opensource.adobe.com/wiki/display/flexsdk/FXG+1.0+Specification
   Accessed: 13th January 2011.

7. Simosol Oy. Simo- Project Code[online].
   URL: http://trac.simo-project.org/browser/simo/trunk/simulator/xml/schemas/parameter_table.xsd
   Accessed: 13th January 2011.

8. Misprintt. Overview of Flex's XML Schema Classes [Online].
URL: http://blog.misprintt.net/?p=192
Accessed: 13th January 2011.

9. Parsley SpiceLib. 20 XML to Object Mapper [Online].
URL:http://www.spicefactory.org/parsley/docs/2.3/manual/?page=xmlmapper&section=intro
Accessed: 13th January 2011.

10. Adobe. Flex 4 Help Document [online].
URL: http://help.adobe.com/en_US/flex/using/flex_4_help.pdf
Accessed: 13th January 2011.

11. Bruce Eckel, James Ward. First Steps in Flex. MindView Inc. CA. 2008.

12. Microsoft (Go Global Development Centre). Understanding Internationalization [Online].
URL: http://msdn.microsoft.com/en-us/goglobal/bb688112.aspx#E1B
Accessed: 13th January 2011.

13. Adobe. Flex 3 (Live Documents) [Online].
URL: http://livedocs.adobe.com/flex/3/html/
Accessed: 13th January 2011.

14. Flex After Dark. Flex Data Providers [Online].
URL: http://www.flexafterdark.com/docs/Flex-DataProviders
Accessed: 13th January 2011.

15. Adobe. Flex in a Week Video Training [Online]. Day 4.
URL: http://www.adobe.com/devnet/flex/videotraining.html

Accessed: 13th January 2011.

16. Hillel Coren. Resources Bundles in Flex [online]. 2008.
    URL: http://hillelcoren.com/2008/09/12/resource-bundles-in-flex-wo-lots-of-extra-code/
    Accessed: 13th January 2011.

17. Chet Haase. States and Components in Flex 4 [Online]. Artima Developer. 2010.
    URL: http://www.artima.com/articles/states_and_components.html
    Accessed: 13th January 2011.

18. Simo Framework documentation [online].
    URL: http://www.simo-project.org/documentation/SIMObook.pdf
    Accessed: 11th March 2011

19. Peter Armstrong Hello Flex. Mnning publication Co. 2010

20. Flex help center: Localization [online]
    URL:http://help.adobe.com/en_US/flex/using/WS2db454920e96a9e51e63e3d11c0bf6119c-7ffd.html
    Accessed: 17th January 2011

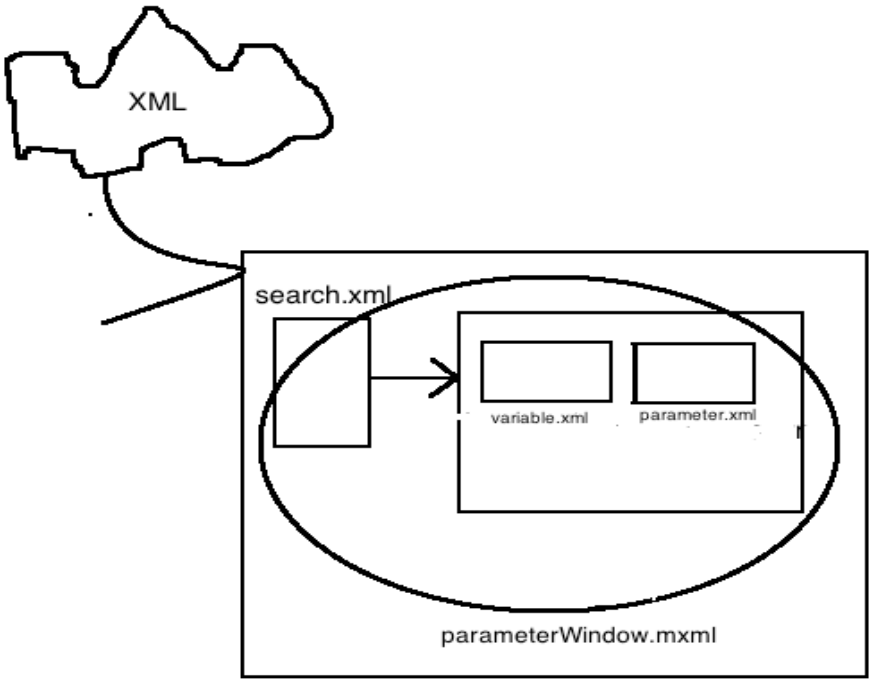
21. Flex help center:Event [online]
    URL:http://help.adobe.com/en_US/flex/using/WS2db454920e96a9e51e63e3d11c0bf64a29-7fff.html
    Accessed: 17th January 2011

22. Learn Flex in a week video tutorial (day 4) [online]
    URL: http://www.adobe.com/devnet/flex/videotraining.html
    Accessed 21st January 2011

24. Beginner's guide to E4X [online]
URL: http://www.nbilyk.com/e4x-example
Accessed 3rd April 2011

Appendices

Appendix 1-The SIMO UIs MECHANISM



Appendix 2- The parameterWindow.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:TitleWindow xmlns:fx="http://ns.adobe.com/mxml/2009"
                        xmlns:s="library://ns.adobe.com/flex/spark"
                        xmlns:mx="library://ns.adobe.com/flex/mx"
                        xmlns:components="components.*"
                        currentState="initial"
                        title="{resourceManager.getString('paramTableUI','title')}"
                        minWidth="780"
                        minHeight="600"
                        width="100%"
                        height="100%"
                        horizontalCenter="0"
                        verticalCenter="0"
                        creationComplete="initsearch(event)"
                        title.initial="Parameter Tables"
                        close="titlewindow_closeHandler()"
                        >
        <fx:Declarations>
                <!-- Place non-visual elements (e.g., services, value objects) here -->
                <fx:XML source="../parameter_table.xml" id="simoXml" format="e4x"/>
                <s:ArrayList
                        id="tableTypes"
                        >
                        <fx:String>operation_parameter</fx:String>
                        <fx:String>data</fx:String>
                </s:ArrayList>
        </fx:Declarations>
        <fx:Metadata>
                [ResourceBundle('paramTableUI')]
        </fx:Metadata>

        <fx:Script source="script.as"/>
        <fx:Script>
                <![CDATA[
                        import mx.collections.ArrayCollection;
                        import mx.collections.XMLListCollection;
                        import mx.controls.Alert;
                        import mx.events.FlexEvent;
                        import mx.managers.PopUpManager;
                        import mx.rpc.events.FaultEvent;
                        import mx.rpc.events.ResultEvent;
                        import mx.rpc.events.XMLLoadEvent;
```

```
import mx.rpc.xml.SimpleXMLDecoder;
import mx.utils.ArrayUtil;
Appendix 2- The parameterWindow.mxml
```

Appendix 2- The parameterWindow.mxml

```
import mx.utils.ObjectUtil;

import skins.openSearchBT;
private var testiteksti:String = "testi";
[Bindable]
public var val:int;
[Bindable]
public var xmlNode:XML = new XML();

protected function addTable_clickHandler(event:MouseEvent):void
{
	const state:String = currentState;
	if (state == 'initial' || state == 'search'){
		currentState='newTable';
	}

}

protected function cancelBT_clickHandler(event:MouseEvent):void
{
	const state:String = currentState;
	if (state == 'newTable'){
		currentState='initial';
	}
}

protected function
openSearchBT_clickHandler(event:MouseEvent):void
{
	const state:String = currentState;
	if (state == 'initial'){
		currentState='search';
	}else currentState='initial';

}


protected function
searchWindow_itemclickHandler(event:SelectedTableEvent):void
```

```
                {
                        removeTable.enabled = true;
                        xmlNode = event.xmlNode;
                        varcomp.displayComp(xmlNode);
                        paramcomp.xmlNode = xmlNode;
```

Appendix 2- The parameterWindow.mxml

```
                varcomp.xmlNodeSelected = true;}
                //updates the classifier datagrid
                protected function refreshDataSheet(event:Event):void{
                        paramcomp.doRefresh;
                }

                protected function titlewindow_closeHandler():void
                {
                        PopUpManager.removePopUp(this);
                }

                private function languageChange(event:Event):void{
                        trace("languageChange");
                        var cb:DropDownList = event.target as DropDownList;
                        var temp:Object = cb.selectedItem;

                        var chain:Array=[];
                        chain.push(temp.locale);
                        for (var i:int=0;i<cb.dataProvider.length;i++ ){
                                if(cb.dataProvider[i] != temp)
                                        chain.push(cb.dataProvider[i].locale);
                        }
                        resourceManager.localeChain = chain;
                }

                private function addXmlTable():void{
                        var xmlString:String =
XmlGenerator.PARAMETER_UI_PARAMETER_TABLE;
                        xmlString = xmlString.replace("$name", tableNameTI.text);
                        xmlString = xmlString.replace("$type",
typeList.selectedItem);
                        xmlString = xmlString.replace("$desc", descTI.text);
                        var newTable:XML = new XML(xmlString);
                        searchWindow.addNode(newTable);
                        const state:String = currentState;
                        if (state == 'newTable'){
                                currentState='initial';
                        }
```

```
                }

                private function removeXmlTable():void{
                        searchWindow.removeNodeAt();
                        if(searchWindow.tableList.dataProvider.length==0)
                                removeTable.enabled=false;}
```

Appendix 2- The parameterWindow.mxml

```
]]>
</fx:Script>

<s:states>
        <s:State
                name="initial"
                />
        <s:State
                name="search"
                />
        <s:State
                name="newTable"
                />
</s:states>

<s:Group id="paramWindow"
                 width="100%"
                 height="100%"
                 minHeight="500"
                 >

        <components:SearchComponent
                id="searchWindow"
                includeIn="initial"
                x="10"
                width="30%"
                xmlList0= "{simocoll}"
                itemclick="searchWindow_itemclickHandler(event)"
                />


        <s:Group id="basicInfo" includeIn="initial" x="250" y="20"  height="496"
width="508">
                <s:DropDownList id="languageCombo"
```

```
        prompt="{resourceManager.getString('paramTableUI','lang')}"
                                                dataProvider="{new
ArrayCollection([{locale:'fi_FI',label:'Suomi'}, {locale:'en_US', label:'English'}])}"
                                                change="languageChange(event)"
                                                width="150" x="330" y="10"/>
                        <s:VGroup
                                width="100%" height="20%"
                                y="10">
```

Appendix 2- The parameterWindow.mxml

```
                                        <s:HGroup gap="10">

                                        <s:Button
                                                id="addTable"

        label="{resourceManager.getString('paramTableUI','newtable')}"
                                                click="addTable_clickHandler(event)"
                                                useHandCursor="true"/>
                                        <s:Button
                                                id="removeTable"

        label="{resourceManager.getString('paramTableUI','removetable')}"
                                                useHandCursor="true"
                                                click="removeXmlTable()"
                                                enabled="false"
                                                />


                                </s:HGroup>
                                <s:Label
                                        id="nameLabel"

        text="{resourceManager.getString('paramTableUI','name')+':
'+xmlNode.attribute('name')}"
                                        />
                                <s:Label
                                        id="typeLabel"

        text="{resourceManager.getString('paramTableUI','type')+':
'+xmlNode.attribute('type')}"
                                        />
                                <s:Label
                                        id="descLabel"
```

```
        text="{resourceManager.getString('paramTableUI','desc')+':
'+xmlNode.attribute('desc')}"
                                    />
                        </s:VGroup>

                        <mx:TabBar
                                id="tabs"
                                dataProvider="{vs}"
                                y="120"
                                tabWidth="240.5"
```

Appendix 2- The parameterWindow.mxml

```
                        paddingLeft="10" paddingRight="10"/>
                                <mx:ViewStack
                                id="vs"
                                width="100%"
                                height="100%"
                                y="141"
                                paddingBottom="20"
                                borderStyle="solid">

                                <s:NavigatorContent

        label="{resourceManager.getString('paramTableUI','vars')}"
                                        width="100%"
                                        height="100%">

                                        <components:VariablesComponent
                                                id="varcomp"
                                                width="100%"
                                                height="100%"
                                                updateClass = "refreshDataSheet(event)"
                                                xmlVal ="{xmlNode}"
                                                simoXml="{simoXml}"
                                                left="10"
                                                right="10"
                                                top="10"
                                                bottom="10"
                                                />

                                </s:NavigatorContent>

                                <s:NavigatorContent
```

```
label="{resourceManager.getString('paramTableUI','title')}"
                                    width="100%"
                                    height="100%">

                                    <components:ParametersComponent
                                        id="paramcomp"
                                        width="100%"
                                        height="100%"
                                        itemCreationPolicy="immediate"
                                        includeIn="initial"
                                        simoXml="{simoXml}"
                                        right="20"
                                        />
```

Appendix 2- The parameterWindow.mxml

```
                        </s:NavigatorContent>
                    </mx:ViewStack>

                </s:Group>

                <s:Group id="addInfo" includeIn="newTable" x="100" y="60">
                    <mx:Form paddingTop="0" paddingLeft="0">
                        <mx:FormItem
label="{resourceManager.getString('paramTableUI','name')}:">
                            <s:TextInput id="tableNameTI" width="200"/>
                        </mx:FormItem>
                        <mx:FormItem
label="{resourceManager.getString('paramTableUI','type')}:">
                            <s:DropDownList id="typeList" width="200"
prompt="{resourceManager.getString('paramTableUI','choose')}"
dataProvider="{tableTypes}"/>
                        </mx:FormItem>
                        <mx:FormItem
label="{resourceManager.getString('paramTableUI','desc')}:">
                            <s:TextInput id="descTI" width="300"/>
                        </mx:FormItem>
                        <mx:FormItem>
                            <s:HGroup paddingTop="10">
                                <s:Button id="saveBT"
label="{resourceManager.getString('paramTableUI','save')}" click="addXmlTable()"/>
                                <s:Button id="cancelBT"
label="{resourceManager.getString('paramTableUI','cancel')}"
click="cancelBT_clickHandler(event)"/>
                            </s:HGroup>
                        </mx:FormItem>
```

```
            </mx:Form>
        </s:Group>

</s:Group>


<s:transitions>
        <s:Transition fromState="initial" toState="newTable">
            <s:Parallel>
                <s:Parallel target="{basicInfo}">
                    <s:Fade duration="100"/>
                </s:Parallel>
```

Appendix 2- The parameterWindow.mxml

```
                <s:Parallel target="{addInfo}">
                    <s:Fade duration="100"/>
                </s:Parallel>
            </s:Parallel>
        </s:Transition>
        <s:Transition fromState="search" toState="newTable">
            <s:Parallel>
                <s:Parallel target="{basicInfo}">
                    <s:Fade duration="100"/>
                </s:Parallel>
                <s:Parallel target="{addInfo}">
                    <s:Fade duration="100"/>
                </s:Parallel>
                <s:Parallel target="{paramWindow}">
                    <s:Resize widthTo="520" duration="300"/>
                </s:Parallel>
            </s:Parallel>
        </s:Transition>
        <s:Transition fromState="newTable" toState="initial">
            <s:Parallel>
                <s:Parallel target="{addInfo}">
                    <s:Fade duration="100"/>
                </s:Parallel>
                <s:Parallel target="{basicInfo}">
                    <s:Fade duration="100"/>
                </s:Parallel>
            </s:Parallel>
```

```
            </s:Transition>
            <s:Transition fromState="initial" toState="search">
                <s:Parallel>
                    <s:Parallel target="{paramWindow}">
                        <s:Resize widthBy="240" duration="300"/>
                    </s:Parallel>
                    <s:Parallel target="{searchWindow}">
                        <s:Fade duration="100" startDelay="300"/>
                    </s:Parallel>
                </s:Parallel>
            </s:Transition>
            <s:Transition fromState="search" toState="initial">
                <s:Parallel>
                    <s:Parallel target="{paramWindow}">
                        <s:Resize widthTo="500" duration="300"
                        startDelay="100"/>
                    </s:Parallel>
```

Appendix 2- The parameterWindow.mxml

```
                    <s:Parallel target="{searchWindow}">
                        <s:Fade duration="100"/>
                    </s:Parallel>
                </s:Parallel>
            </s:Transition>
        </s:transitions>

</s:TitleWindow>
```